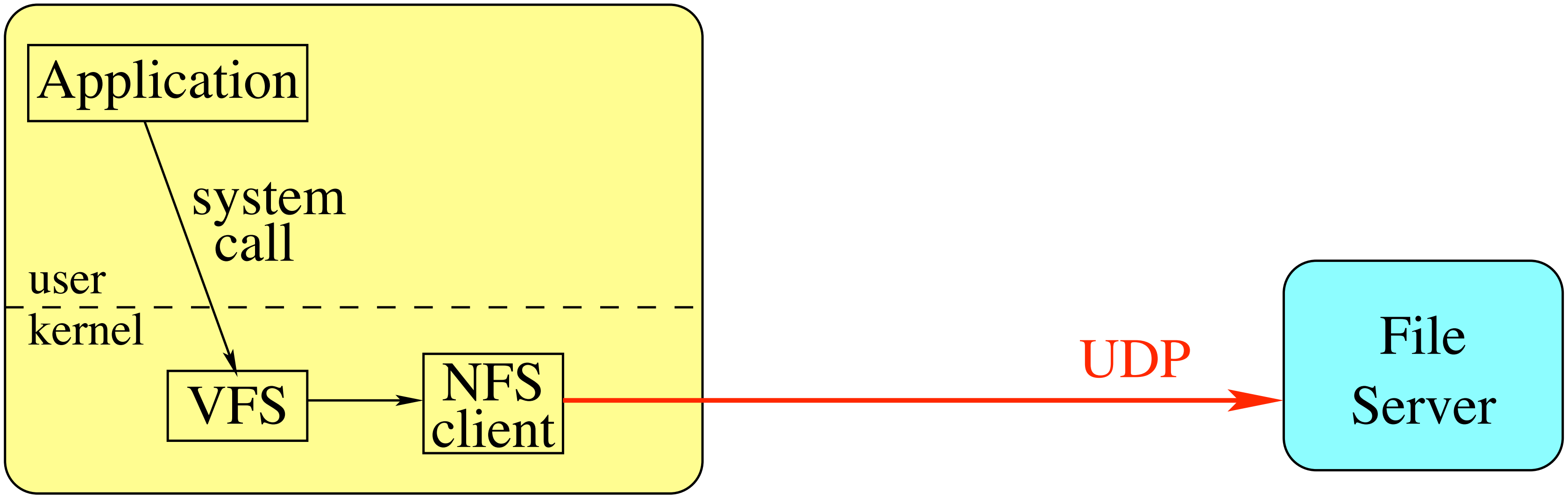
User-level file systems

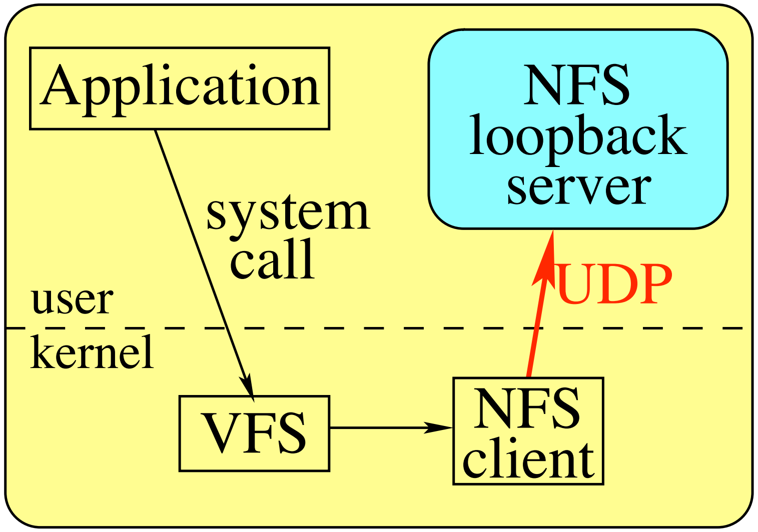
- **Developing new file systems is a difficult task**
  - Most file systems implemented in the kernel
  - Debugging harder, crash/reboot cycle longer
  - Complicated kernel-internal API (VFS layer)
- **File systems are not portable**
  - Kernel VFS layer differs significantly between OS versions
- **NFS can solve these problems…**
  - C++ toolkit greatly simplifies the use of NFS

NFS overview



- **NFS is available for almost all Unixes**
- **Translates file system accesses into network RPCs**
  - Hides complex, non-portable VFS interface

Old idea: NFS loopback servers



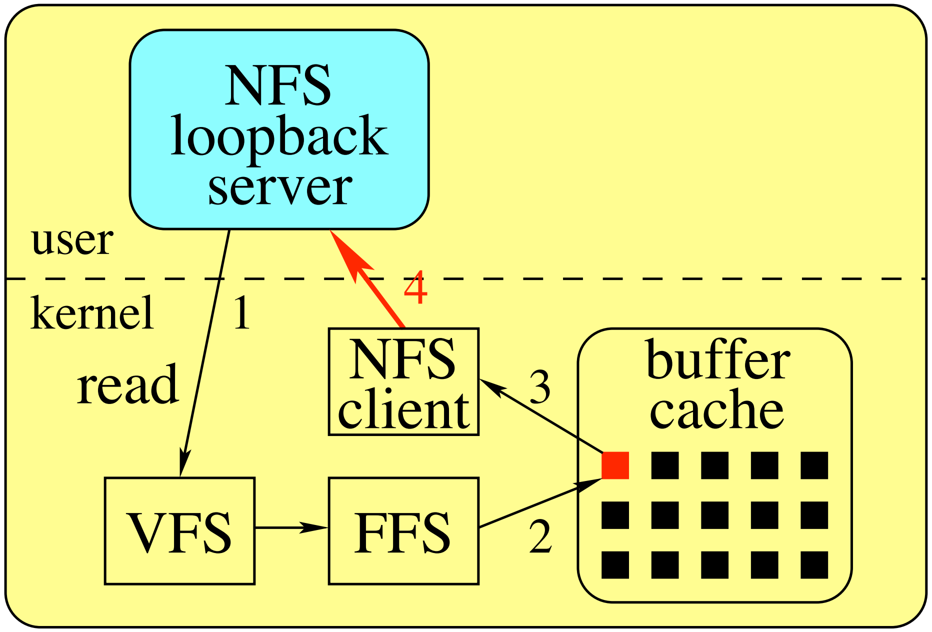
- **Implement FS as an NFS server in a local process**
- **Requires only portable, user-level networking**
  - File system will run on any OS with NFS support

Problem: Performance

- **Context switches add latency to NFS RPCs**
- **Must service NFS RPCs in parallel**
  - Overlap latencies associated with handling requests
  - Keep disk queue full for good disk arm scheduling
- **If loopback server blocks, so do other processes**
  - E.g., loopback for `/loop` blocks on a TCP connect
  - *getcwd()* and “`ls -al /`” will block, even outside of `/loop`
- **One slow file can spoil the whole file system** [a]
  - If one RPC times out, client decides server is down
  - Client holds other RPCs to avoid flooding server
  - Example: Alex FTP file server

[a] `NFS3ERR_JUKEBOX` can help, but has problems

Problem: Any file I/O can cause deadlock



1. Loopback server reads file on local disk
2. FFS needs to allocate a buffer
3. Kernel chooses a dirty NFS buffer to recycle
4. Blocks waiting for reply to write RPC

Problem: Development and debugging

- **Bugs must be mapped onto NFS RPCs**
  - Application make system calls
  - Not always obvious what RPCs the NFS client will generate
  - Bug may actually be in kernel's NFS client
- **When loopback servers crash, they hang machines!**
  - Processes accessing the file system hang, piling up
  - Even `umount` command accesses the file system and hangs
- **Repetitive code is very error-prone**
  - Often want to do something for all 20 NFS RPC procedures (e.g., encrypt all NFS file handles)
  - Traditionally requires similar code in 20 places

SFS toolkit

- **Goal: Easy construction of loopback file systems**
- **Support complex programs that never block**
  - Service new NFS RPCs while others are pending
- **Support multiple mount points**
  - Loopback server emulates multiple NFS servers
  - One slow mount point doesn’t hurt performance of others
- **Simplify task of developing/debugging servers**
  - `nfsmounter` daemon eliminates hangs after crashes
  - RPC library supports tracing/pretty-printing of NFS traffic
  - RPC compiler allows traversal of NFS call/reply structures

`nfsmounter` daemon

- **`nfsmounter` mounts NFS loopback servers**
  - Handles OS-specific details of creating NFS mount points
  - Eliminates hung machines after loopback server crashes
- **To create an NFS mount point, loopback server:**
  - Allocates a network socket to use for NFS
  - Connects to `nfsmounter` daemon
  - Passes `nfsmounter` a copy of the NFS socket
- **If loopback server crashes:**
  - `nfsmounter` takes over NFS socket
  - Prevents processes accessing file system from blocking
  - Serves enough of file system to unmount it

Asynchronous I/O and RPC libraries

- **Never wait for I/O or RPC calls to complete**
  - Functions launching I/O must return before I/O completes
  - Bundle up state to resume execution at event completion
- **Such event-driven programming hard in C/C++**
  - Cumbersome to bundle up state in explicit structures
  - Often unclear who must free allocated memory when
- **Alleviated by two C++ template hacks**
  - `wrap`—function currying: bundles function of arbitrary signature with initial arguments
  - Reference counted garbage collection for any type:
    `ptr<T> tp = new refcounted<T> (/* ...  */);`

`rpcc`: A new RPC compiler for C++

- **Compiles RFC1832 XDR types to C++ structures**
  - Saw native representations last lecture
- **Produces generic code to traverse data structures**
  - RPC marshaling only one possible application
- **Can specialize traversal to process particular types**
  - Encrypt/decrypt all NFS file handles for security
  - Extract all file attributes for enhanced caching
- **Outputs pretty-printing code**
  - ASRV_TRACE, ACLNT_TRACE environment variables make library print all RPC traffic
  - Invaluable for debugging strange behavior

Stackable NFS manipulators

- **Often want to reuse/compose NFS processing code**
- **SFS toolkit provides stackable NFS manipulators**
  - NFS server objects generate NFS calls
  - Most loopback servers begin with `nfsserv_udp`
  - Manipulators are servers constructed from other servers
- **Example uses:**
  - `nfsserv_fixup`—works around bugs in NFS clients
  - `nfsdemux`—demultiplex requests for multiple mount points

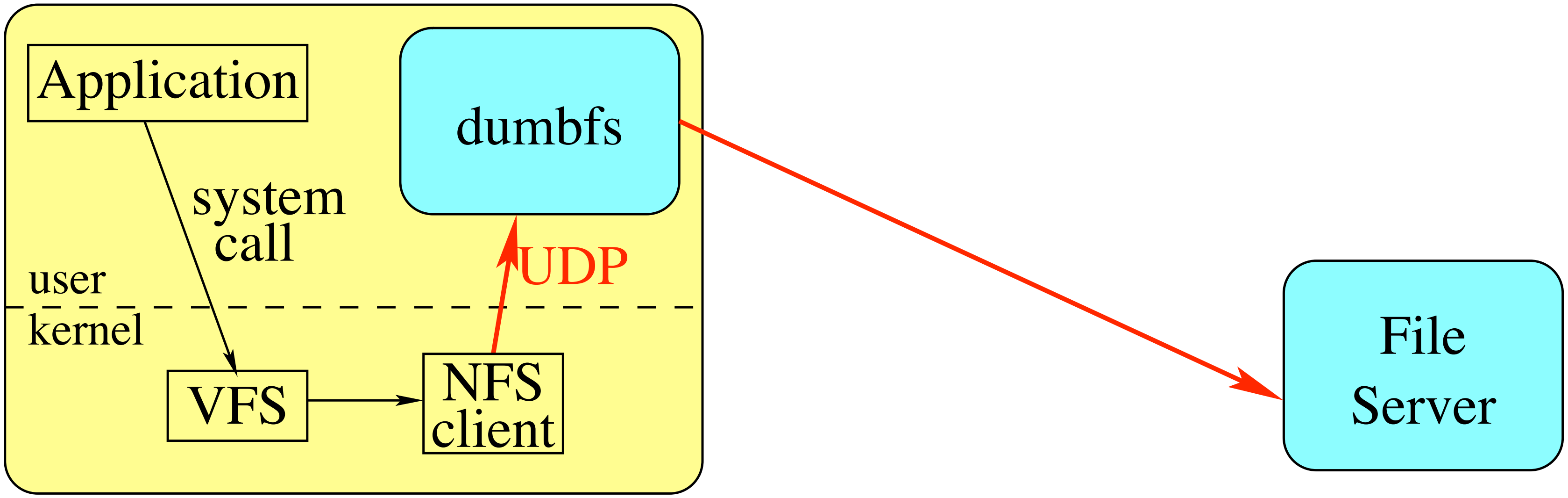
Creating new mountpoints

- **Hard to create mountpoints in-place and on-the-fly**
  - If user looks up `/home/u1`, must reply before mounting
  - Previous loopback servers use links: `/home/u1`→`/a/srv/u1`
- **SFS automounter mounts in place with two tricks**
  - `nfsmounter` has special gid, differentiating its NFS RPCs
  - SFS dedicates "wait" mountpoints under `.mnt/`{`0,1,...`}
- **Idea: Show different files to users and** `nfsmounter`
  - User sees `/home/u1` as symlink `u1`→`.mnt/0/0`
  - `.mnt/0/0` is symlink that hangs when read
  - `nfsmounter` sees `/home/u1` as directory, can mount there
  - When mount complete, `.mnt/0/0`→`/home/u1`

Limitations of loopback servers

- **No file close information**
  - Often, FS implementor wants to know when a file is closed (e.g., for close-to-open consistency of shared files)
  - Approximate "close simulator" exists as NFS manipulator
  - NFS version 4 will include closes
- **Can never delay NFS writes for local file system**
  - E.g., CODA-like cache hard to implement

Application: DumbFS



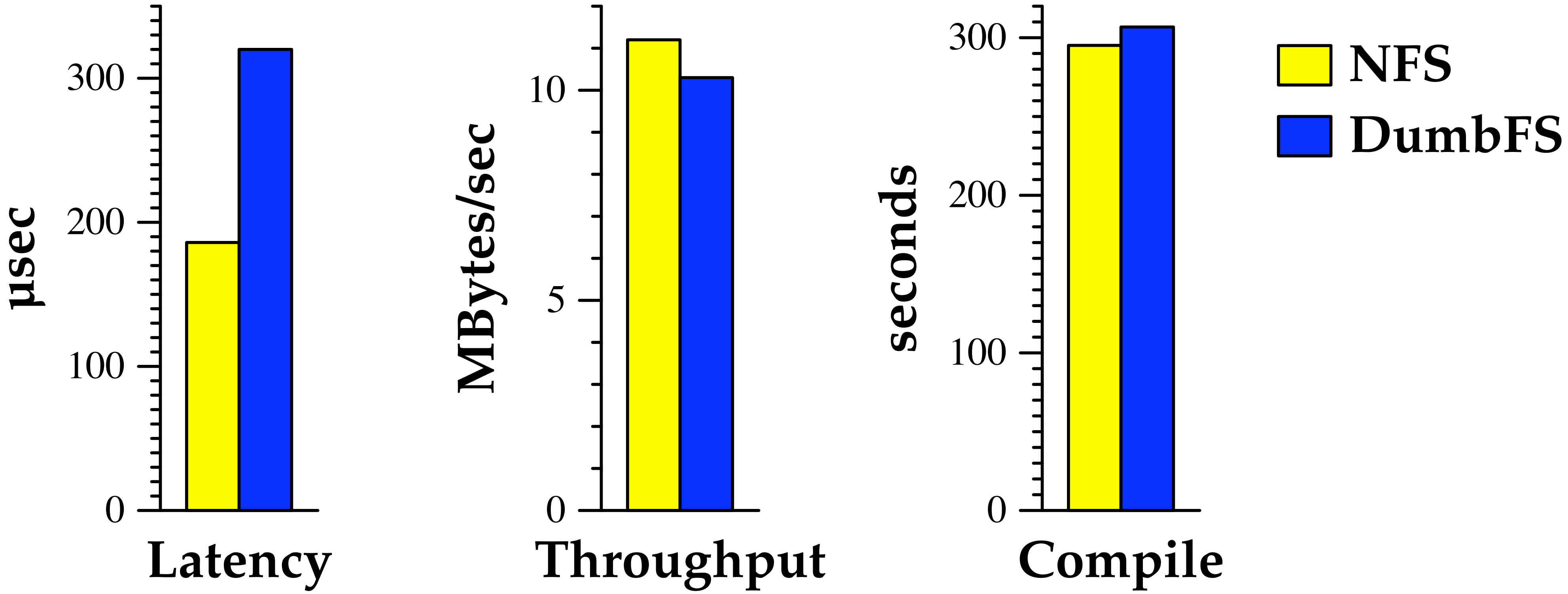
- **Simplest loopback server—just forwards requests**
  - 119 lines of code, no cleanup code needed!
- **Isolates performance impact of toolkit**

DumbFS NFS RPC forwarding

```
void dispatch (nfscall *nc)
{ // ...
  nfsc->call (nc->proc (), nc->getvoidarg (),
              nc->getvoidres (), wrap (reply, nc) /* ... */);
}
static void reply (nfscall *nc, enum clnt_stat stat)
{
  if (stat == RPC_SUCCESS) nc->reply (nc->getvoidres ());
  else // ...
}
```

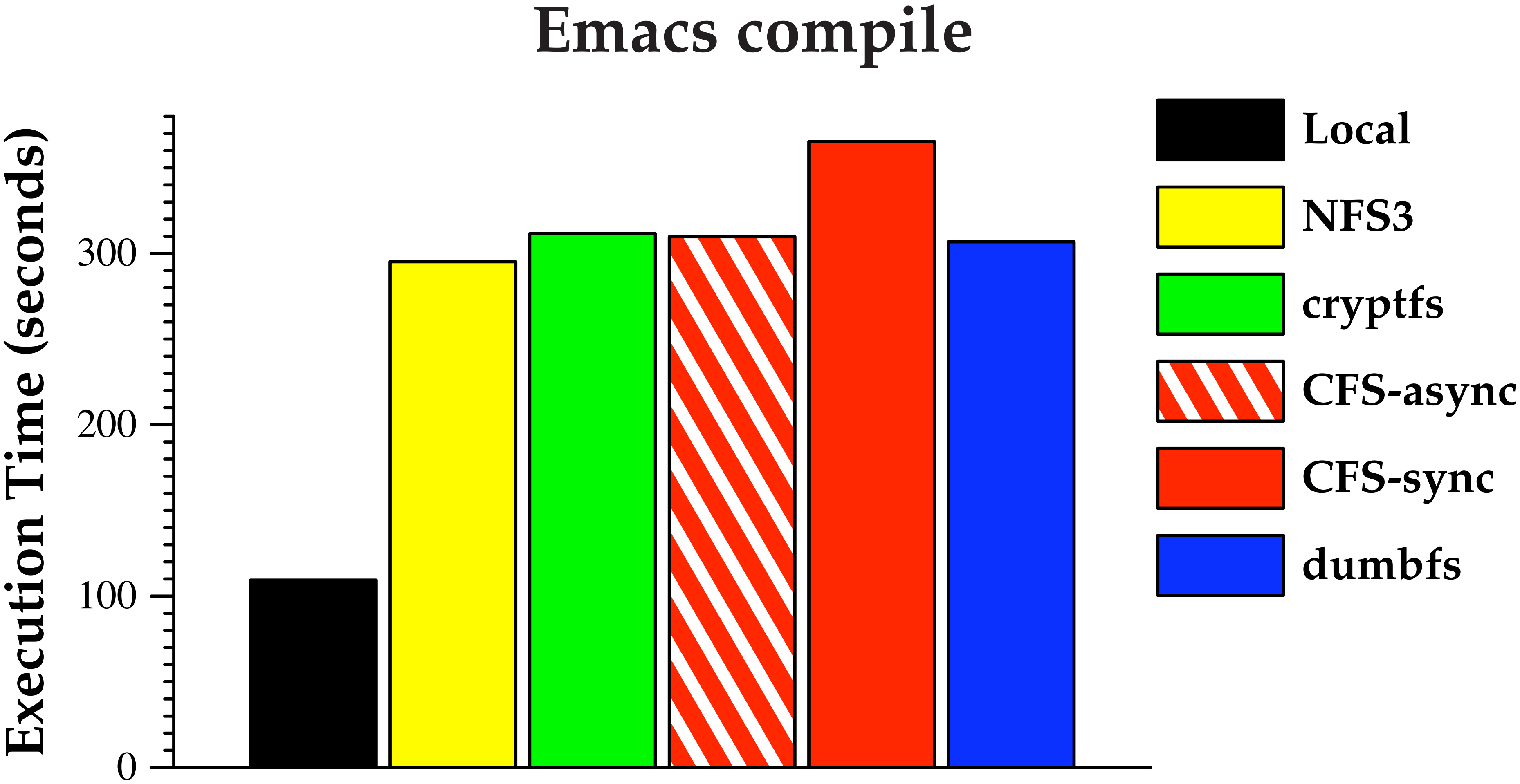
- **Single dispatch routine for all NFS procedures**
- **RPCs to remote NFS server made asynchronously**
  - `dispatch` returns before `reply` invoked

DumbFS performance



Application: CryptFS

- **Acts as both NFS server and client (like DumbFS)**
  - Almost 1–1 mapping between NFS calls recevied and sent …encrypt/decrypt file names and data before relaying
  - Bare bones "encrypting DumbFS" <1,000 lines of code, Complete, usable system <2,000 lines of code
- **Must manipulate call/reply of 20 RPC proceedures**
  - Encrypted files slightly larger, must adjust size in replies
  - All 20 RPC procedures can contain one more file sizes
  - RPC library lets CryptFS adjust 20 return types in 15 lines


Emacs compile
Execution Time (seconds)
0
100
200
300
Local
NFS3
cryptfs
CFS-async
CFS-sync
dumbfs

User-level FS summary

- **NFS allows portable, user-level file systems**
  - Translates non-portable VFS interface to standard protocol
- **In practice, loopback servers have had problems**
  - Low performance, blocked processes, deadlock, debugging difficulties, redundant, error-prone code,…
- **SFS toolkit makes most problems easy to avoid**
  - `nfsmounter` eliminates hangs after crashes
  - `libasync` supports complex programs that never block
  - `rpcc` allows concise manipulation of 20 call/return types
  - Stackable manipulators provide reusable NFS processing